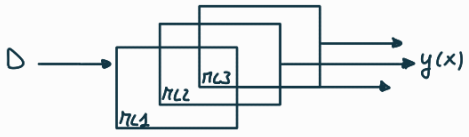
Multiple learners

The main idea of multiple learners is to split the computation in order to use the computational budget to train different models instead of just one.

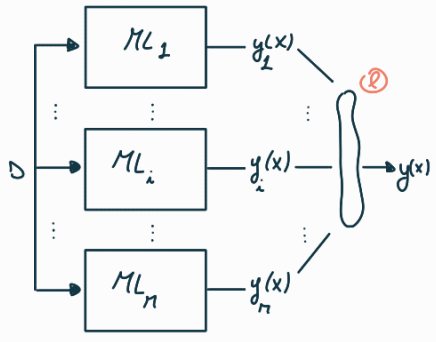
Before

Now - ensemble learning



We have 3 approaches: ① voting, ② bagging, ③ boosting } Parallel learning: each learner can be trained independently from the others

① Voting: the same dataset is used to train multiple models in parallel; each model is trained independently; we can have either the same model with different parameters or completely different models. All the results are then combined. The criterion to combine the results can be fixed (majority voting, weighted combination, ...) or can be trainable (the layer ② can be trainable).
The main problem with this approach is that there may be low variability between the models and sometimes the outputs $y_1(x), \dots, y_M(x)$ can be very similar. We must hope that the ensembled output is better than the single ones. Sharing the same dataset is not a good option.



① Use $\mathcal{D}$ to train a set of models $y_m(x)$ for $m = 1, \dots, M$
② Make predictions with:

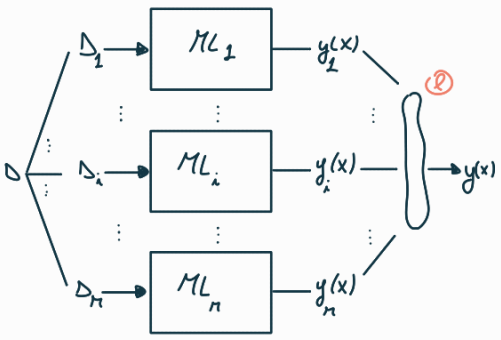
$$y_{voting}(x) = \sum_{m=1}^{M} w_m y_m(x) \qquad \text{for regression}$$

$$y_{voting}(x) = \underset{c}{\text{argmax}} \sum_{m=1}^{M} w_m I(y_m(x) = c) \qquad \text{for classification}$$

$I(e) = 1$ if e is True, 0 otherwise

this is called weighted majority: every model will vote for one class, you count how many votes each class has received and you can optionally give a weight to the votes

② **Bagging**: very similar to voting but we first preprocess the original dataset and split it in different parts, one for each model. Each dataset is a subset of the original one (not necessary a partition, there could be repetitions). We have better variability (less bias due to the dataset).



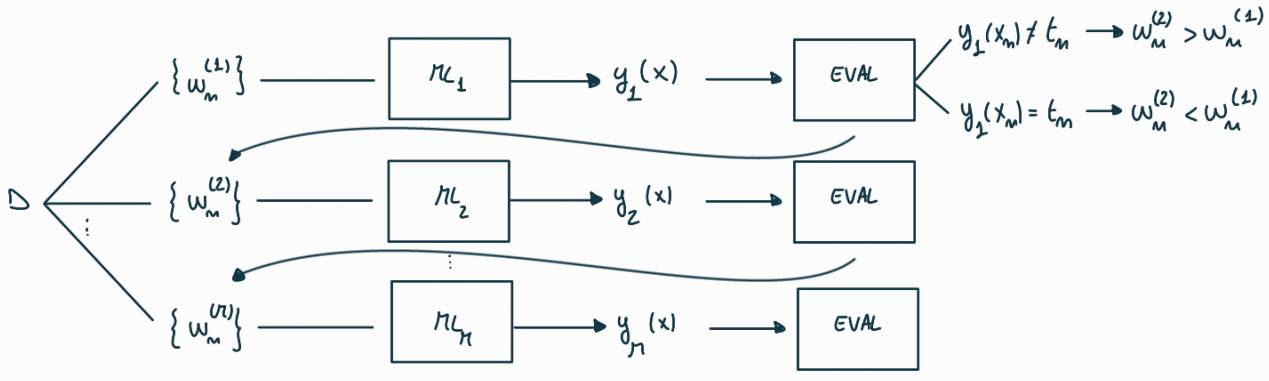
① Generate M bootstrap data sets $D_1, \dots, D_M$ with $D_m \subset D$
② Use each bootstrap data set D_m to train a model $y_m(x)$ for $m = 1, \dots, M$
③ Make predictions with a voting scheme

$$y_{Bagging}(x) = \frac{1}{M} \sum_{m=1}^{M} y_m(x)$$

In general this is better than training any individual model.
Bootstrap data sets are chosen with random sampling with replacement.

The difference between voting and bagging is in how the dataset is used
Voting ⟶ same dataset for each model
Bagging ⟶ different dataset for each model

③ Boosting: we have a sequential learning; idea: the next layer should be better on samples that are not correctly classified by the previous ones. We associate a weight to each (x_n, t_n) in the dataset; after a model is trained we evaluate it; if the evaluation fails $y(x) \neq t_i$ we increase the weight of that sample for the next model. If the evaluation is correct we decrease the weight. At the beginning each sample has the same weight $1/N$.



When we train a model we define a loss/error function $E(\dots;\theta)$ (depending on the parameters of the model θ). The error function will give a measure of the difference between the predicted value and the true one.

$$E(\dots;\theta) = \dots (y(x_i) - t_i)$$

Considering a weighted dataset, we just add the weight to the error function, multiplying it for the difference.

$$E(\dots;\theta) = \dots w_i (y(x_i) - t_i)$$

The error will be bigger for samples with high weight. This way we introduce in the error function the importance of each sample.

Main idea: simple models quickly reach reasonable, but not good, performances in little time; instead of having a single model, we use multiple simple models and combine them. We will surely get a higher accuracy with respect to the one that we will obtain training the bigger model for the same amount of time

Summary

- Instead of designing a learning algorithm that is accurate over the entire space, one can focus on finding base learning algorithms that only need to be better than random
- Combined learners theoretically outperforms any individual learner
- AdaBoost practically outperforms many other base learners in many problems.
- Ensembles of small Deep NN outperform very deep NN in some cases; Small Deep NN are faster to train and require less computational resources.

AdaBoost

Given $\mathcal{D} = \{(x_1, t_1), \dots, (x_N, t_N)\}$, where $x_n \in \mathcal{X}, t_n \in \{-1, +1\}$

① Initialize $w_n^{(1)} = 1/N$, $n = 1, \dots, N$ — at the beginning each sample has the same weight

② For $m = 1, \dots, M$:

① Train a weak learner $y_m(x)$ by minimizing the weighted error function :

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n) \quad \text{with } I(e) = \begin{cases} 1 & \text{if } e \text{ is True} \\ 0 & \text{otherwise} \end{cases}$$

$w_n^{(m)}$: learner m

② Evaluate

$$\alpha_m = \ln\left(\frac{1 - \epsilon_m}{\epsilon_m}\right)$$

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$$

total weights corresponding to the errors

total weights

③ Update the data weighting coefficients

$$w_n^{(m+1)} = w_n^{(m)} e^{\alpha_m I(y_m(x_n) \neq t_n)}$$

we update the weight for the next learner. When $e = y_m(x_n) == t_n$ is True there is no mistake on this sample and the weigh won't be changed $w_n^{(m+1)} = w_n^{(m)} e^0 = w_n^{(m)}$

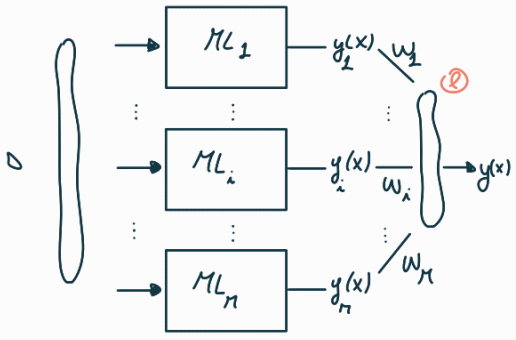
If the model at layer m makes a mistake on sample n then $e = y_m(x_n) == t_n$ is False and $w_n^{(m+1)} = w_n^{(m)} e^{\alpha_m}$

③ Output of the final model :

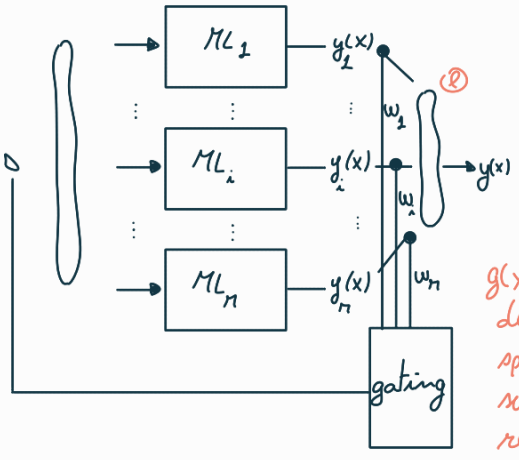
$$Y_M(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(x)\right)$$

We can have variations of these methods with respect to how the final output is computed.

- classical voting ⟶ weighted average (as before)



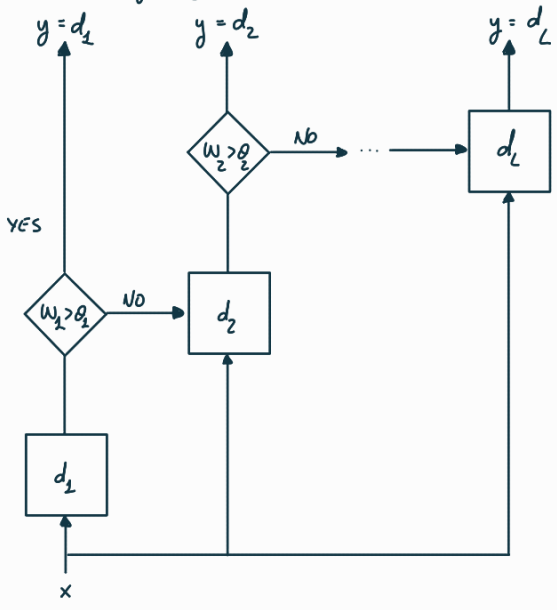
- gating function ⟶ weights are not predefined but depends on the input



$g(x)$
depending on the region of the input space some learners are more suitable to predict that input with respect to others

this is not always easy but this procedure can be automatically learned

- stacking ⟶ automatically learn the gating function
- cascading ⟶ learners based on confidence thresholds; you still train the weights of a gating function but then, when you classify a new instance, instead of combining all the results you use a cascading approach: you first try the first learner, if you are satisfied by the confidence threshold you use that result, otherwise you go to the next learner.



We don't have a merging behaviour we have the output of exactly one learner; which learner depends on the confidence.